Fast enumeration algorithms for non-crossing geometric graphs

AUTHOR(S):

Katoh, Naoki; Tanigawa, Shin-Ichi

Fast Enumeration Algorithms for Non-crossing Geometric Graphs

Naoki Katoh and Shin-ichi Tanigawa

Department of Architecture and Architectural Engineering, Kyoto University,
Kyoto Daigaku Katsura, Nishikyo-ku, Kyoto 615-8540 Japan,
{naoki,is.tanigawa}@archi.kyoto-u.ac.jp

Abstract

A non-crossing geometric graph is a graph embedded on a set of points in the plane with non-crossing straight line segments. In this paper we present a general framework for enumerating non-crossing geometric graphs on a given point set. Applying our idea to specific enumeration problems, we obtain faster algorithms for enumerating plane straight-line graphs, non-crossing spanning connected graphs, non-crossing spanning trees, and non-crossing minimally rigid graphs. Our idea also produces efficient enumeration algorithms for other graph classes, for which no algorithm has been reported so far, such as non-crossing matchings, non-crossing red-and-blue matchings, non-crossing k-vertex, or k-edge connected graphs or non-crossing directed spanning trees. The proposed idea is relatively simple and potentially applies to various other problems of non-crossing geometric graphs.

1 Introduction

Given a graph $G = (V, E)$ with n vertices and m edges where $V = \{1, \ldots, n\}$, an embedding of the graph on a set of points $P = \{p_1, \cdots, p_n\} \subset \mathbb{R}^2$ is a mapping of the vertices to the points in the Euclidean plane $i \mapsto p_i$. A *geometric graph* is a graph embedded on P such that each edge (i, j) of G is mapped to a straight line segment (p_i, p_j). A set of embedded segments is called *non-crossing* if any pair of elements does not have a point in common except possibly their endpoints, and a geometric graph is called non-crossing if its corresponding straight line segments are non-crossing.

In this paper we assume that a given point set P is *fixed* in $\mathbb{R}^2$ and an embedding $V \to P$ is given. Since a *graph class* is defined in terms of the properties that all its members share, imposing the additional "non-crossing" requirement to an existing graph class, we can define a *non-crossing geometric graph class* on P, such as non-crossing spanning trees or non-crossing perfect matchings. Let us denote by $\mathcal{NGG}$ a specific non-crossing geometric graph class. We shall extensively study the following *enumeration problem*:

Input: A point set P in the plane with n points.
Output: The list of all the non-crossing geometric graphs belonging to $\mathcal{NGG}$ on P.

Since the output of the problem may consist of exponentially many graphs in terms of the input size, the efficiency of the *enumeration algorithm* is measured customarily in both the input and output sizes. In particular, if the computational time can be bounded by a polynomial in the input size and by a linear function in the output, the algorithm is said to work in *polynomial time (on average)*.

In this paper we present a new general framework for enumerating non-crossing geometric graphs. Our new framework provides faster algorithms for various enumeration problems compared with existing ones, such as those for *plane straight-line graphs*, *non-crossing spanning connected graphs*, *non-crossing spanning trees*, and *non-crossing minimally rigid graphs*. Moreover, since the idea is quite simple, it can be applied to many enumeration problems, for which enumeration algorithms were not known to the best of our knowledge, such as *non-crossing matchings*, *non-crossing red-and-blue matchings*, *non-crossing k-vertex or k-edge connected graphs* or *non-crossing directed geometric*

Table 1: Time complexities of new algorithms and previous ones.

	New results	Previous best results
plane straight-line graphs	$O(\mathsf{pg}(P))$	$O(n \log n \cdot \mathsf{pg}(P))$ [2]
non-crossing spanning connected graphs	$O(\mathsf{cg}(P))$	$O(n \log n \cdot \mathsf{cg}(P))$ [2]
non-crossing spanning trees	$O(n \cdot \mathsf{tri}(P) + \mathsf{st}(P))$	$O(n \log n \cdot \mathsf{st}(P))$ [2]
non-crossing minimally rigid graphs	$O(n^2 \cdot \mathsf{mrf}(P))$	$O(n^3 \cdot \mathsf{mrf}(P))$ [8, 9]
non-crossing perfect matchings	$O(n^{3/2} \cdot \mathsf{tri}(P) + n^{5/2}\mathsf{pm}(P))$	—

graphs. In Table 1 we list the time complexities of (a part of) new algorithms obtained in this paper, where we use the following notation to denote the numbers of graphs on a point set P: $\mathsf{pg}(P)$ for plane straight-line graphs, $\mathsf{cg}(P)$ for non-crossing spanning connected graphs, $\mathsf{st}(P)$ for non-crossing spanning trees, $\mathsf{mrf}(P)$ for non-crossing minimally rigid graphs, $\mathsf{tri}(P)$ for triangulations, and $\mathsf{pm}(P)$ for non-crossing perfect matchings.

The key idea of our technique is to use triangulations. Let us consider enumerating all non-crossing spanning trees for example. Since every subgraph of a triangulation is non-crossing, enumerating all non-crossing spanning trees in a triangulation is easily done by applying algorithms such as [19, 32] developed for enumerating all spanning trees in a given (abstract) graph. Moreover, efficient enumeration algorithms for triangulations are already known [7, 12]. Therefore, by enumerating spanning trees in every triangulation, we will obtain all non-crossing spanning trees since every non-crossing spanning tree is a subgraph of some triangulation. However, some non-crossing spanning tree might be produced more than once since it could be a subgraph of more than one triangulation.

For some specific graph classes, Avis et al. [9] and the authors [21] have shown how to avoid duplicate generation based on a well-known enumeration framework, called the *reverse search* [6, 7]. In this paper we extend this idea and develop a new general technique which does not rely on the property of a particular graph class. In order to avoid duplicate enumeration we introduce two key notions: *edge-constrained lexicographically largest triangulations* (which were originally introduced in [21] for the development of an efficient enumeration algorithm of edge-constrained non-crossing spanning trees) and *minimal representative sets*. For a set of non-crossing segments F, a geometric graph containing F is called *F-constrained*. We will show that, for each triangulation T, there exists the inclusionwise minimum non-crossing edge set F^*, called the minimal representative set, such that T is the F^*-constrained lexicographically largest triangulation (that is the triangulation of the lexicographically largest edge list among all F^*-constrained triangulations with respect to a certain order on edges defined later). In the enumeration algorithm proposed in this paper, every time a new triangulation T is obtained, we will compute the minimal representative set F^* of T and then enumerate all spanning trees that are contained in T and *contain* F^* as the subset. We will show that this algorithm correctly enumerates all non-crossing spanning trees without repetitions.

The overall idea of our techniques will be described in two algorithms, Algorithm 1 and Algorithm 2, in Sections 3 and 4, respectively. Let $\mathsf{ngg}(P)$ be the total number of graphs of $\mathcal{NGG}$ to be enumerated. Then, Algorithm 1 enumerates all the non-crossing geometric graphs belonging to $\mathcal{NGG}$ without repetitions in $O(f(n) \cdot \mathsf{tri}(P) + g(n) \cdot \mathsf{ngg}(P))$ time, where f is a polynomial function, and g is a function of n depending on $\mathcal{NGG}$. In the graph classes considered in this paper, g is also a polynomial function. By applying Algorithm 1 we obtain new algorithms for enumerating plane straight-line graphs, non-crossing spanning connected graphs, non-crossing spanning trees and non-crossing perfect matchings (see Table 1). In particular, for plane straight-line graphs or non-crossing spanning connected graphs, we show that $\mathsf{pg}(P)$ and $\mathsf{cg}(P)$ are exponentially larger than $\mathsf{tri}(P)$ for *every* point set P, and thus the term of $f(n) \cdot \mathsf{tri}(P)$ is dominated by $\mathsf{pg}(P)$ or $\mathsf{cg}(P)$. Consequently, our algorithms work in $g(n)$ time on average, which will be shown to be constant. These results improve the running time of the previous best ones by Aichholzer et al. [2].

Although Algorithm 1 enumerates all graphs of $\mathcal{NGG}$ efficiently in terms of $\mathsf{tri}(P)$ and $\mathsf{ngg}(P)$, its time complexity cannot be bounded by $O(g(n) \cdot \mathsf{ngg}(P))$ in general since its complexity is dom-

inated by $\mathsf{tri}(P)$ when $\mathsf{tri}(P)$ is much larger than $\mathsf{ngg}(P)$. The next proposed algorithm Algorithm 2 overcomes this drawback by avoiding the enumeration of the triangulations T which contain no F^*-constrained geometric graph of $\mathcal{NGG}$ for the minimal representative set F^* of T. Applying Algorithm 2, we obtain an enumeration algorithm for non-crossing minimally rigid graphs that works in $O(n^2)$ time on average. This result improves the previous one by Avis et al. [8] by an $O(n)$ factor on average.

As for related work of our paper, Welzl [36] recently showed a relatively similar approach for counting the total number of planar straight-line graphs on a given point set, where he proposed the method of using the edge-constrained Delaunay triangulation and so-called Lawson edges, which in our context correspond to the edge-constrained lexicographically largest triangulation and the minimal representative set, respectively. We remark that our work has been done independently from it. In addition, as the current fastest enumeration algorithm [12] for triangulations is based on the lexicographical ordering on the edge set, there are some advantages of using the lexicographically largest triangulation over the Delaunay triangulation especially in the time complexity analysis (e.g., a simple amortized analysis of the edge insertion algorithm given in Section 4). Also, the concept of the lexicographically largest triangulation enables us to prove a nontrivial lower bound on the number of non-crossing spanning connected graphs, which will be given in Theorem 3.7.

Enumerating combinatorial objects is a fundamental problem, and several algorithms have been developed for non-crossing geometric graphs, e.g., triangulations [7, 12], non-crossing spanning trees [2, 7, 21], pseudo-triangulations [10, 13], and non-crossing minimally rigid graphs [8, 9]. Let us explain why the enumeration of non-crossing geometric graphs is more difficult than that of non-geometric (abstract) graphs. The *branch-and-bound technique* (or sometimes called the binary-partition technique, see, e.g., [33, 34]) is a well-known framework for designing enumeration algorithms. Consider, for example, the problem for enumerating all spanning trees in a (multi)graph G with n vertices and m edges. Then, we can easily design an algorithm that enumerates all spanning trees in $O(m^2)$ time per output graph as follows. The algorithm repeatedly divides the problem into two subproblems: one enumerates the spanning trees containing an edge e of G, and the other enumerates those not containing e. In the first subproblem, e is contracted (and resulting loops are removed if there exists any), while in the second subproblem, e is removed. Then, the problem size is surely reduced in each subproblem. Moreover, since it can be checked in $O(m)$ time whether the resulting graph contains at least one spanning tree, the algorithm can decide correctly whether it should continue the search or not. Therefore, by going down this branch-and-bound tree in $O(m)$ steps, the algorithm surely detects a new spanning tree.

The branch-and-bound technique provides us with polynomial-time enumeration algorithms for many graph classes because it just requires a polynomial-time oracle that checks whether a given graph contains at least one subgraph belonging to a certain graph class. However, the problem of detecting a non-crossing subgraph in a given geometric graph is known to be NP-hard for most graph classes (even in the case of non-crossing spanning trees or non-crossing perfect matchings [26]). For this reason, most of the enumeration problems for non-crossing geometric graphs become nontrivial and we need to introduce some new technique. In fact, all previous works for the enumeration of non-crossing geometric graphs are based not on the branch-and-bound technique but on sophisticated local transformations discussed below.

Two objects of $\mathcal{NGG}$ are *connected* if they can be transformed into each other by a *transformation* which generates one graph from the other by a certain specified operation. Define a graph $\mathcal{G}_{\mathcal{NGG}}$ on $\mathcal{NGG}$ where the vertex set of $\mathcal{G}_{\mathcal{NGG}}$ corresponds to the set of all objects of $\mathcal{NGG}$ and two vertices are connected by an edge if the corresponding graphs of $\mathcal{NGG}$ are connected. Then, the natural question is how we can design a transformation so that $\mathcal{G}_{\mathcal{NGG}}$ is a connected graph. Moreover, from the viewpoint of the applications to enumeration problems, a transformation should be defined *locally*, i.e., the symmetric difference between two connected objects should be as small as possible. Developing such a nice transformation might be interesting in its own right, and there are many known results not only for *local transformations* [2, 5, 7, 21–23, 25] but also for *large transformations* [1, 3, 24]. Almost all previous works for the enumerations of non-crossing geometric graphs discussed above are based

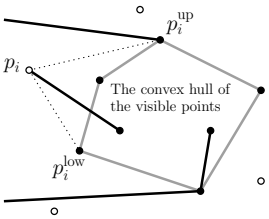


Figure 1: An example of the upper and lower tangents, denoted by (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$, respectively. The bold edges represent F.

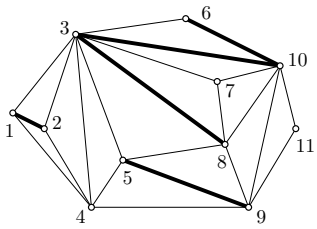


Figure 2: F-CLLT.

on local transformations, and thereby they deeply rely on the property of a particular graph class. On the other hand, the proposed technique in this paper reveals that efficient enumeration of $\mathcal{NGG}$ is possible without defining a local transformation of $\mathcal{NGG}$ explicitly.

2 The Edge-Constrained Lexicographically Largest Triangulation

Recall that a geometric graph containing a set of non-crossing segments F is called *F-constrained*. In this section, we first introduce some notation used throughout the paper and then provide a number of preliminary results on the F-constrained lexicographically largest triangulation (F-CLLT). These results were originally obtained in [21] for the purpose of developing an efficient enumeration algorithm for edge-constrained non-crossing spanning trees and play a crucial role in the development of our framework.

2.1 Notation

Let P be a set of n points in $\mathbb{R}^2$, and for simplicity we label the points $P = \{p_1, \ldots, p_n\}$ in the increasing order of x-coordinates. We assume that the x-coordinates of all points are distinct and that no three points of P are collinear. For two points $p_i, p_j \in P$, we use the notation $p_i < p_j$ if $i < j$ holds, and $p_i = p_j$ if they coincide. Considering $p_i \in P$, we often pay attention only to the point set to its right, $\{p_{i+1}, \ldots, p_n\} \subseteq P$, which is denoted by P_{i+1}.

Let K_n be the complete graph embedded on P (with straight line segments). The line segment between p_i and p_j with $p_i < p_j$ is called *edge*, denoted by (p_i, p_j). We often consider a geometric graph G as an *edge set* and use the notation G to denote the edge set of G for simplicity when it is clear from the context.

For three points p_i, p_j and p_k, the signed area $\Delta(p_i, p_j, p_k)$ of the triangle $p_i p_j p_k$ tells us whether p_k is on the left (or right, resp.) side of a line passing through p_i and p_j when moving along the line from p_i to p_j by $\Delta(p_i, p_j, p_k) > 0$ (or $\Delta(p_i, p_j, p_k) < 0$, respectively). We define a total ordering $\prec$ on the set of edges as follows: for $e = (p_i, p_j)$ and $e' = (p_k, p_l)$, $e \prec e'$ holds if $p_i < p_k$, or $p_i = p_k$ and $\Delta(p_i, p_j, p_l) < 0$. Notice that the ordering of e and e' is determined by the clockwise ordering around p_i if $p_i = p_k$. Let $E = \{e_1 \prec \cdots \prec e_m\}$ and $E' = \{e'_1 \prec \cdots \prec e'_m\}$ be sorted edge lists in increasing ordering. Then, E' is *lexicographically larger* than E if $e_i \prec e'_i$ for the smallest i such that $e_i \neq e'_i$.

We say that two edges (p_i, p_j) and (p_k, p_l) *properly intersect* if (p_i, p_j) and (p_k, p_l) have a point in common except for their endpoints. For two points $p_i, p_j \in P$ and a non-crossing edge set F, we say that p_j is *visible from p_i with respect to F* when the edge (p_i, p_j) does not properly intersect any edge of F, but we assume that p_j is *visible* from p_i if $(p_i, p_j) \in F$.

Upper and *lower tangents*, (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$, *of p_i with respect to F* are defined as the supporting edges from p_i to the convex hull of the points of P_{i+1} that are visible from p_i with respect to F (see Fig. 1).

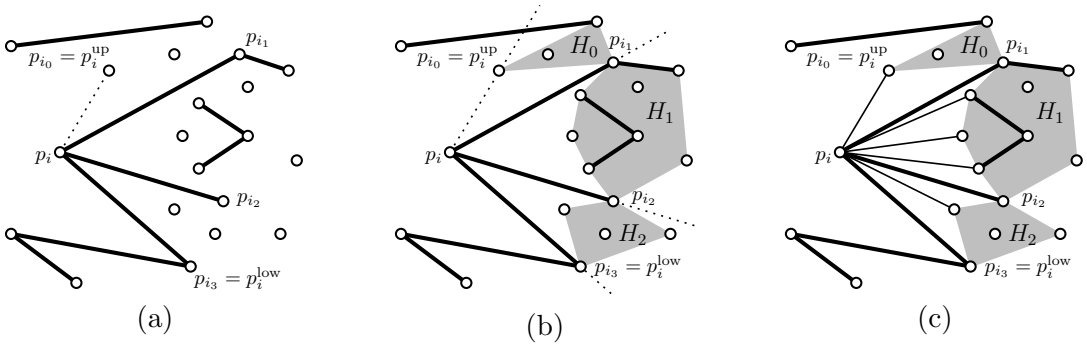


Figure 3: Construction 1 around p_i where the bold edges represent F. (a)Step 1, (b)Step 2 and (c)Step 3.

2.2 The Edge-constrained Lexicographically Largest Triangulations

For a non-crossing edge set F on P and a point $p_i \in P$, let us denote by $\delta_F(p_i)$ the set of edges of F whose left endpoints are p_i. Let us consider the following construction of an F-constrained geometric graph on P:

Construction 1.

0. *Repeat the following process for all $p_i \in P$ in an arbitrary order.*

1. *Let (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$ be the upper and lower tangents of $p_i \in P$ with respect to F, and denote the right endpoints of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ by $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ arranged in clockwise order around p_i (where $p_{i_0} = p_i^{\mathrm{up}}$ and $p_{i_m} = p_i^{\mathrm{low}}$ hold) (Fig.3(a)).*

2. *Consider the cone C_k with apex at p_i bounded by two consecutive edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$ for each k with $0 \leq k \leq m-1$, where C_k contains both p_{i_k} and $p_{i_{k+1}}$, and construct the convex hull H_k of $P_{i+1} \cap C_k$ inside each C_k (Fig.3(b)).*

3. *Draw an edge from p_i to every point $p_j \in P_{i+1} \cap C_k$ such that $p_j = (p_i, p_j) \cap H_k$ for each k (Fig.3(c)).*

We give an example of the graph obtained by the above construction in Fig. 2. Notice that the graph obtained by Construction 1 always has the edges of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ for all $p_i \in P$. The following property has been proved in [21].

Proposition 2.1. ([21]) *The graph G obtained by Construction 1 is an F-constrained triangulation on P. Moreover, it has the lexicographically largest edge list among all F-constrained triangulations on P.*

Hence, we call the F-constrained triangulation obtained by the above construction the *F-constrained lexicographically largest triangulation* (F-CLLT). Although the details are omitted, we can show that the F-CLLT can be also constructed by greedily adding the edges of K_n to F in the descending order with respect to $\prec$ without violating the non-crossing property. Note that the F-constrained lexicographically largest triangulation is uniquely determined for every F since $\prec$ is a total ordering over the edges of K_n.

An edge e in a triangulation T is called *flippable* if the two triangles incident to e in T form a *convex* quadrilateral Q. *Flipping* e in T generates a new triangulation by replacing e with the other diagonal of Q. In [21], we showed that every F-constrained triangulation can be transformed into F-CLLT by flipping $O(n^2)$ edges not in F, each of which increases the lexicographical order of the edge list.

2.3 Deleting and Inserting the Constrained Edge

Let $\mathcal{F}$ be the collection of all non-crossing edge sets on a given point set P, and let $\mathcal{T}$ be the collection of all triangulations on P. We will often treat a triangulation as an *edge set* in the

subsequent discussion. We make use of the construction of the F-CLLT as a function $T^* : \mathcal{F} \to \mathcal{T}$ that maps a non-crossing edge set F to the corresponding F-CLLT $T^*(F)$. The following properties of the function T^* are crucial for developing the general technique in the next section (a part of them has been given in [21]). We shall provide proofs for completeness.

Lemma 2.2. *Let $F \in \mathcal{F}$. Then, for every $e \in T^*(F)$, $T^*(F \cup \{e\}) = T^*(F)$ holds.*

Proof. Suppose, for a contradiction, that there is $e \in T^*(F)$ such that $T^*(F \cup \{e\}) \neq T^*(F)$. Since $F \cup \{e\} \subseteq T^*(F)$, the triangulation $T^*(F \cup \{e\})$ has a lexicographically larger edge list than that of $T^*(T^*(F)) = T^*(F)$. This is a contradiction because $F \subseteq F \cup \{e\}$ implies that $T^*(F)$ has a lexicographically larger edge list than $T^*(F \cup \{e\})$. □

Lemma 2.3. *Let $F \in \mathcal{F}$. Then, for $E \subseteq F$, $T^*(F \setminus E) = T^*(F)$ holds if and only if every $e = (p_i, p_j) \in E$ is (i) the upper or lower tangent of p_i with respect to F or (ii) non-flippable in $T^*(F)$.*

Proof. ("Only-if" part:) Assume, for a contradiction, that there exists $e = (p_i, p_j) \in E$ satisfying neither (i) nor (ii) of the statement when $T^*(F \setminus E) = T^*(F)$ holds. Notice that $T^*(F \setminus E) = T^*(F)$ implies that $T^*(F)$ is and $(F \setminus E)$-constrained lexicographically largest triangulation and an F-constrained lexicographically largest triangulation. Consider the two triangles of $T^*(F \setminus E)$ incident to e and denote the two vertices appearing in these triangles other than p_i and p_j by v and w. Since e is flippable in $T^*(F \setminus E)(= T^*(F))$, the quadrilateral $p_i v p_j w$ is convex. In addition, since e is neither upper nor lower tangent of p_i, both v and w lie on the right side of p_i, and hence $e \prec (v, w)$ holds. Therefore, flipping e to (v, w) produces an $(F \setminus E)$-constrained triangulation that is lexicographically larger than $T^*(F)$, which is a contradiction.

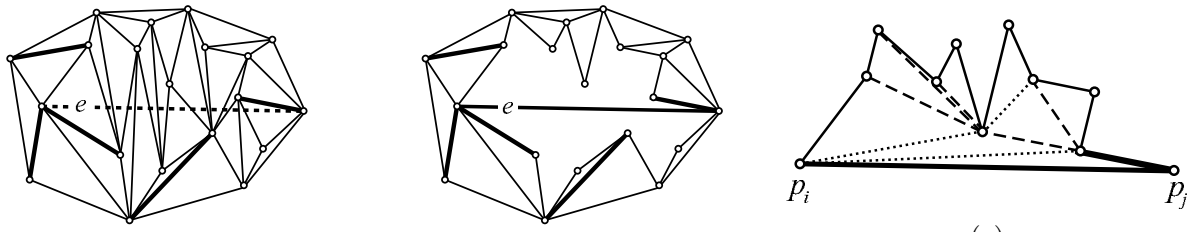
("If" part:) We shall show that, for $e = (p_i, p_j) \in E$, $T^*(F \setminus \{e\}) = T^*(F)$ holds if e satisfies (i) or (ii) of the statement. If so, since each edge of $E \setminus \{e\}$ still satisfies (i) and (ii) in $T^*(F \setminus \{e\}) = T^*(F)$, removing the edges of E one by one from the constraint, we eventually obtain $T^*(F \setminus E) = T^*(F)$.

First let us consider the case where e satisfies (i). Since removing $e = (p_i, p_j)$ does not affect the visibility of p_i, e is also the upper or lower tangent of p_i with respect to $F \setminus \{e\}$. Since $T^*(F \setminus \{e\})$ contains every tangent from the definition of Construction 1, $e \in T^*(F \setminus \{e\})$ follows, implying $T^*(F \setminus \{e\}) = T^*(F)$ from Lemma 2.2.

Next let us consider the case where e satisfies (ii). We can assume that e is neither upper nor lower tangent. We shall show that e is still contained in $T^*(F \setminus \{e\})$, which in turn implies $T^*(F \setminus \{e\}) = T^*(F)$. Let (p_i, p_i^{up}) and $(p_i, p_i^{\mathrm{low}})$ be the upper and lower tangents of p_i with respect to F. Since removing $e = (p_i, p_j)$ does not affect the visibility of p_i as mentioned above, they are, respectively, the upper and lower tangents of p_i with respect to $F \setminus \{e\}$.

Let us denote the set of the right endpoints of $\delta_F(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ by $p_{i_0}, p_{i_1}, \ldots, p_{i_m}$ in clockwise ordering around p_i. Since $e \in \delta_F(p_i)$ and e is neither upper nor lower tangent, $e = (p_i, p_{i_k})$ holds for some k with $1 \leq k \leq m-1$. According to the definition of Construction 1, there exists a convex chain between p_{i_k} and $p_{i_{k+1}}$ in $T^*(F)$ such that the face bounded by this convex chain and the two edges (p_i, p_{i_k}) and $(p_i, p_{i_{k+1}})$ form a pseudo-triangle, which contains no point of P in its interior. Similarly, there exists the convex chain between $p_{i_{k-1}}$ and p_{i_k} in $T^*(F)$ and the corresponding pseudo-triangle. Since e is non-flippable in $T^*(F)$, combining these two pseudo-triangles, we obtain a single pseudo-triangle which consists of the convex chain from $p_{i_{k-1}}$ to $p_{i_{k+1}}$ and the two edges $(p_i, p_{i_{k-1}})$ and $(p_i, p_{i_{k+1}})$. Let us denote this pseudo-triangle by t. Note that t contains no point of P.

Suppose, for a contradiction, that $e \notin T^*(F \setminus \{e\})$ holds. Then, there exists an edge $e' \in T^*(F \setminus \{e\})$ such that e' properly intersects e. However, considering the pseudo-triangle t, e' must intersect at least one of $(p_i, p_{i_{k-1}})$ and $(p_i, p_{i_{k+1}})$ because $e = (p_i, p_{i_k})$ is contained in (the face of) t and the three convex corners of t are $p_i, p_{i_{k-1}}$, and $p_{i_{k+1}}$. Therefore, $T^*(F \setminus \{e\})$ misses at least one of $(p_i, p_{i_{k-1}})$ and $(p_i, p_{i_{k+1}})$, which contradicts that $T^*(F \setminus \{e\})$ contains all edges of $\delta_{F \setminus \{e\}}(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$, as we remarked right after the description of Construction 1. This is a contradiction. □



(a) (b) (c)

Figure 4: (a)Insertion of a new constrained edge e. (b)The two empty simple polygons obtained by removing the edges I properly intersecting e. (c)Reconstruction inside the polygon in the upper side, where the dashed and dotted edges represent the type (1) and type (2) edges.

2.4 Maintaining the F-constrained Lexicographically Largest Triangulation

Let us discuss how to maintain the F-CLLT when we newly insert one constrained edge e to F. Developing the following efficient way to construct $T^*(F \cup \{e\})$ from $T^*(F)$ will be helpful for constructing the fast enumeration algorithm discussed in Section 4.1.

Lemma 2.4. *Let $T^*(F)$ be the F-CLLT on a given set of n points, and let e be an edge that does not properly intersect any edge of F. Then, it takes $O(n)$ time to construct $T^*(F \cup \{e\})$ from $T^*(F)$.*

Proof. Let $e = (p_i, p_j)$, and let I be the set of edges of $T^*(F)$ that properly intersect e. Let us first verify the following fact: Every edge of $T^*(F) \setminus I$, say $(p_k, p_l) \in T^*(F) \setminus I$, is still contained in $T^*(F \cup \{e\})$.

Let us consider how $T^*(F)$ is determined by Construction 1 around p_k. Since $(p_k, p_l) \in T^*(F)$, there exists the cone C_F with apex p_k considered in Step 2 of Construction 1 which contains p_l, and the convex hull H_F of $P_{k+1} \cap C_F$ with $p_l = (p_k, p_l) \cap H_F$. Similarly, when constructing $T^*(F \cup \{e\})$, in Step 2 we shall consider the convex hull H_{F+e} inside some cone with apex p_k such that $p_l \in H_{F+e}$.

When inserting e, the vertices that are not visible from p_k with respect to F remain nonvisible from p_k with respect to $F \cup \{e\}$. This implies $H_{F+e} \subseteq H_F$. Hence, $p_l = (p_k, p_l) \cap H_F$ implies $p_l = (p_k, p_l) \cap H_{F+e}$, and (p_k, p_l) remains in $T^*(F \cup \{e\})$ from Construction 1.

Therefore, the update occurs only inside the two polygons obtained by removing the edges of I and adding e (see Figures 4(a) and 4(b)). Without loss of generality, we assume that e is horizontal, and let us show an efficient algorithm to triangulate (the interior of) the polygon lying on the upper side of $e = (p_i, p_j)$ (the lower side can be treated similarly). Consider the updated triangulation of the polygon by Construction 1. There exist two types of new edges: (1) lower tangent of each vertex of the polygon with respect to the boundary edges of the polygon, and (2) the others (see Figure 4(c)). We call them type (1) and type (2), respectively.

Let us consider how to find the type (1) edges. Let v be a vertex of the polygon which misses the lower tangent in $(T^*(F) \setminus I) \cup \{e\}$, that is, e properly intersects the lower tangent (v, v^{low}) of v with respect to F which existed in $T^*(F)$. Consider a ray emanating from v to v^{low}, which first hits e before reaching v^{low}. Rotating the ray around v in counterclockwise order inside the polygon until it encounters a vertex of the polygon, we can find the new lower tangent $(v, \tilde{v}^{\mathrm{low}})$ of v, which is a type (1) edge (if $(v, \tilde{v}^{\mathrm{low}})$ does not already exist in $T^*(F)$). We repeatedly continue the rotation of the ray around the newly encountered vertex $\tilde{v}^{\mathrm{low}}$ of the polygon until the ray encounters p_j. Since we are rotating the ray in one direction, the sequence of vertices encountered in this process induces a convex chain connecting p_j and some vertices of the polygon. Consequently, the set of all type (1) edges is a subset of the convex chains connecting p_j and each vertex of the polygon as shown in Figure 4(c), which represent the shortest paths inside the polygon from p_j. It is known [18] that the shortest paths from a single source to all vertices inside a simple polygon can be computed in $O(n)$ time, although it requires an involved linear-time algorithm for triangulating a simple polygon [14]. Thus, we could obtain the desired time complexity through the shortest path algorithm.

Our problem, however, can be solved easily by performing a Graham scan (see, e.g., [15]) only once. Let us try to construct the lower part of the convex hull of the vertices of the polygon by performing a Graham scan algorithm from p_j to p_i. We remark that the algorithm scans all vertices not in the order of the coordinates as usual but in the vertex sequence order of the polygon from p_j to p_i. When we encounter a new vertex p during the scan, we examine the top vertex q and the next one r on the stack. If the angle of the three points around q inside the polygon is convex (i.e. $\Delta(r, q, p) > 0$), we draw the edge between p and r and then pop q from the stack. We continue this process until we obtain three vertices p, q' and r' whose angle around q' inside the polygon is reflex, that is, $\Delta(r', q', p) < 0$. Then, we insert p into the stack and proceed to the next vertex. Repeating this process until $p = p_i$ and the stack contains only p_i and p_j, we can draw all of the required edges in linear time. □

3 Enumerating Non-crossing Geometric Graphs

3.1 General Idea

We define the relation $\sim$ on $\mathcal{F}$ as follows; for two non-crossing edge sets F and F', $F \sim F'$ if and only if $T^*(F) = T^*(F')$. Let $[T] = \{F \in \mathcal{F} \mid F \sim T\}$ for each $T \in \mathcal{T}$.

Lemma 3.1. *The relation $\sim$ is an equivalence relation on $\mathcal{F}$. The collection $\{[T] \mid T \in \mathcal{T}\}$ of all equivalence classes forms a partition of $\mathcal{F}$.*

Proof. Since $T^*(F)$ is uniquely determined for each $F \in \mathcal{F}$, $\sim$ clearly is an equivalence relation. Hence, $[T]$ is an equivalence class and thus $\{[T] \mid T \in \mathcal{T}\}$ forms a partition of $\mathcal{F}$. □

We say that an edge e of $F \in \mathcal{F}$ is the *smallest* or *largest* one among F if it is the smallest edge, or respectively the largest edge, among F with respect to the edge ordering $\prec$. We remark that the upper tangent (and the lower tangent, resp.) of p_i with respect to F is the smallest edge (and the largest edge, resp.) in $\{(p_i, q) \in T^*(F) \mid q \in \{p_{i+1}, \ldots, p_n\} = P_{i+1}\}$. This implies that, for any $F \in [T]$ of a triangulation T, the upper and lower tangents with respect to F are equivalent to the smallest and largest ones of $\{(p_i, q) \in T \mid q \in P_{i+1}\}$. Using Lemma 2.3, a unique minimal representative set for each $[T]$ is defined as follows.

Lemma 3.2. *Let T be a triangulation on a given point set P, and let F^* be the set of all flippable edges in T except for the smallest and largest edges of $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ for every $p_i \in P$. Then,*

(i) $F^* \in [T]$ (i.e., $T^*(F^*) = T$), and

(ii) for any $F \in \mathcal{F}$, $F \in [T]$ if and only if $F^* \subseteq F \subseteq T$.

Proof. Let us show (i). It is obvious that $T^*(T) = T$. Note that, by the definition of F^*, every edge $e = (p_i, p_j) \in T \setminus F^*$ is non-flippable in T, or the smallest or largest edge among $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ (i.e., e is the upper or lower tangent of p_i with respect to T). Hence, by Lemma 2.3, removing $T \setminus F^*$ does not change the triangulation, that is, $T = T^*(T) = T^*(T \setminus (T \setminus F^*)) = T^*(F^*)$.

Next, let us show (ii). The "if-part" can be proved in the same way as in the first part. In fact, removing the edges of $F \setminus F^*$, we obtain $T^*(F) = T^*(F \setminus (F \setminus F^*)) = T^*(F^*) = T$ by Lemma 2.3. Let us consider the "only-if" part. It is obvious that $F \subseteq T$ if $F \in [T]$. Suppose that F (with $F \subseteq T$) is a counterexample, that is, $T^*(F) = T$ but $F^* \setminus F \neq \emptyset$. Then an edge $e = (p_i, p_j) \in F^* \setminus F$ is flippable in T and neither the smallest nor largest edge among $\{(p_i, q) \in T \mid q \in P_{i+1}\}$ by the definition of F^*, which implies $T = T^*(F) = T^*(F^* \setminus (F^* \setminus F)) \neq T^*(F^*)$ by Lemma 2.3. This contradicts $T = T^*(F^*)$. □

Thus, we call F^* defined in Lemma 3.2 the *minimal representative set* of T, denoted by $R(T)$. Our enumeration algorithm, which consists of two phases, can be easily described as follows.

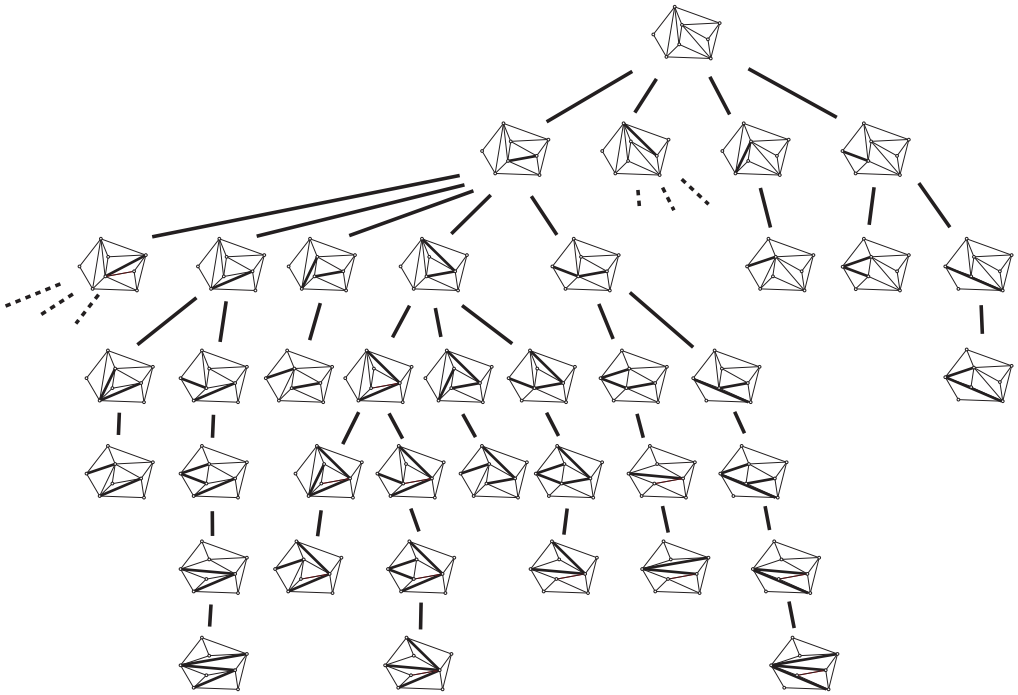


Figure 5: The search tree on the set of triangulations obtained by the algorithm by Bespamyatnikh, where each minimal representative set is drawn in bold.

Algorithm 1: Enumeration of $\mathcal{NGG}$.

Phase 1: Enumerate all triangulations for a given point set P based on the fast enumeration algorithm by Bespamyatnikh [12].

Phase 2: Every time a new triangulation T is found, enumerate all graphs G contained in T such that $G \in \mathcal{NGG}$ and G contains the minimal representative set $R(T)$ as its subset, i.e., G is an $R(T)$-constrained graph in T.

Let $\mathcal{C}$ be the graph class obtained by relaxing the non-crossing constraint from the non-crossing geometric graph class $\mathcal{NGG}$ (i.e., the collection of geometric graphs whose edge sets are not necessarily non-crossing but satisfy the combinatorial properties of $\mathcal{NGG}$). Notice that in Phase 2 the problem for enumerating all graphs of $\mathcal{NGG}$ is reduced to that of enumerating all elements of $\mathcal{C}$ containing $R(T)$ in a triangulation T because T is non-crossing. This implies that we may utilize an oracle for enumerating all the graphs of $\mathcal{C}$ in a given (abstract) graph and we can ignore "geometric" and "non-crossing".

The algorithm needs $R(T)$ explicitly for every T in Phase 2, and hence it will be better to maintain and update $R(T)$ during the enumeration of triangulations rather than to compute it from scratch. The task of Phase 1 is in fact not only the enumeration of T but also the generation of $R(T)$. This additional task can be handled by slightly modifying the triangulation enumeration, which will be discussed more formally in Section 3.2. Figure 5 shows an example of the enumeration of triangulations and the minimal representative sets.

Theorem 3.3. *Algorithm 1 enumerates all graphs of $\mathcal{NGG}$ without repetitions.*

Proof. Consider an arbitrary graph $G \in \mathcal{NGG}$. Then, $T = T^*(G)$ is uniquely determined. This implies $G \in [T]$ and $G \notin [T']$ for any triangulation T' with $T' \neq T$ by Lemma 3.1. Since $G \in [T]$ implies $R(T) \subseteq G \subseteq T$ by Lemma 3.2, Phase 2 of Algorithm 1 for the triangulation T enumerates G by an (assumed) oracle. On the other hand, $G \notin [T']$ implies $R(T') \not\subseteq G$ or $G \not\subseteq T'$. Thus, any G is enumerated exactly once in Phase 2 for $T = T^*(G)$. □

3.2 Time Complexity of Algorithm 1

In order to analyze the time complexity of Algorithm 1, let us briefly review the enumeration algorithm of triangulations by Bespamyatnikh [12], which is based on the reverse search technique [7]. The reverse search is a well-known technique to generate all elements of the considered combinatorial objects by tracing the nodes in the *search graph*, in which a node corresponds to an object to be enumerated and an edge corresponds to a *transformation* (discussed in the introduction) between two objects. To trace the search graph efficiently, the algorithm defines a *root* node and a unique *parent* for each node except for the root such that the subgraph of the search graph induced by the parent-child relations forms a rooted spanning tree. Such a spanning tree is called *search tree*, and the algorithm traces it by depth-first manner. The search graph of the algorithm by Bespamyatnikh is defined in such a way that two triangulations are connected if and only if they can be transformed to each other by a diagonal flip (see Fig. 5 or [12] for more details). The following lemma states how to efficiently maintain the minimal representative set during the enumeration of triangulations.

Lemma 3.4. *Let T_1 and T_2 be two triangulations for which T_2 is obtained from T_1 by a diagonal flip of the edge f. Then, the size of the symmetric difference between $R(T_1)$ and $R(T_2)$ is constant. More specifically, only the four edges of the two triangle faces incident to f are involved in the symmetric difference.*

Proof. Let us first characterize $e \in R(T_1) \setminus R(T_2)$ with $e \neq f$. There are two cases: (Case 1) $e = (p_i, p_j) \in R(T_1)$ becomes non-flippable in T_2, and (Case 2) $e = (p_i, p_j)$ becomes the smallest or largest edge among $\{(p_i, q) \in T_2 \mid q \in P_{i+1}\}$ in T_2. Notice that a diagonal flip switches at most the four flippable edges in T_1 into non-flippable edges in T_2, and hence, if e is an edge of Case 1, it must be one of the four edges of the triangles incident to f. Let us consider Case 2. Since e is not the smallest (or largest, resp.) one in $\{(p_i, q) \in T_1 \mid q \in P_{i+1}\}$, there exists an edge $e' = (p_i, q')$ in T_1 with $e' \prec e$ (or $e \prec e'$, resp.) such that e' and e are incident to a common triangle face of T_1. We notice that e' disappears in T_2 because e becomes the smallest (or largest) one, and hence e' is exactly f. So, f and e are incident to the same triangle face in T_1.

The analogous argument works for $e \in R(T_2) \setminus R(T_1)$. □

By Lemma 3.4, during Algorithm 1, the symmetric difference of the minimal representative sets can be output in $O(1)$ time if the triangulation is maintained in a proper data structure and a flag is attached to each edge to indicate whether it is in the minimal representative set or not. The algorithm by Bespamyatnikh [12] enumerates all triangulations in $O(\log \log n)$ time per output. Thus, we obtain the following theorem:

Theorem 3.5. *Let $\mathcal{C}$ be the graph class obtained by relaxing the non-crossing constraint from $\mathcal{NGG}$. Suppose that there exists an algorithm for enumerating all $R(T)$-constrained graphs of $\mathcal{C}$ in a triangulation without repetitions in time $t_{\mathcal{C}}$ per output graph with preprocessing time $t_{\mathcal{C},\mathrm{pre}}$. Then, all graphs of $\mathcal{NGG}$ on a given point set P of n points can be enumerated without repetitions in $O((\log \log n + t_{\mathcal{C},\mathrm{pre}}) \cdot \mathsf{tri}(P) + t_{\mathcal{C}} \cdot \mathsf{ngg}(P))$ time, where $\mathsf{tri}(P)$ and $\mathsf{ngg}(P)$ are the total numbers of triangulations and $\mathcal{NGG}$ on P, respectively.*

Most of the enumeration algorithms we will use as a subroutine in the applications take $t_{\mathcal{C},\mathrm{pre}} = O(n)$ time in the preprocess phases (see Section 3.3).

3.3 Applications of Algorithm 1

3.3.1 Enumerating Non-crossing Spanning Trees

We show here how to apply Algorithm 1 to the enumeration of non-crossing spanning trees on a given point set. What we have to consider here is just how to enumerate all spanning trees in a given triangulation T, each of which contains the minimal representative set $R(T)$. We remark again that, in the above process, we do not have to care about whether an output spanning tree is non-crossing

because T is non-crossing. In Phase 2 of Algorithm 1, we use the algorithm for enumerating all spanning trees on a given undirected graph developed by Kapoor and Ramesh [19] or Shioura et al. [31, 32]. These algorithms can enumerate all spanning trees of a given graph in $O(1)$ time per output graph[1] with $O(n+m)$ preprocessing time, where n and m denote the numbers of vertices and edges of a given graph. The edge constraint can be handled easily by contracting the constraint edges before calling these oracles. Hence, applying algorithms of [19, 31, 32] to the resulting (multi-)graph, we can enumerate all the $R(T)$-constrained spanning trees contained in T in $t_{\mathcal{C}} = O(1)$ time per output graph with $t_{\mathcal{C},\mathrm{pre}} = O(n)$ preprocessing time (for contracting the edges of $R(T)$ and for the preprocessing of [19, 31, 32]). Thus, by Theorem 3.5 the following result is derived:

Theorem 3.6. *Let P be a set of n points in the plane. Then the set of non-crossing spanning trees on P can be enumerated in $O(n \cdot \mathsf{tri}(P) + \mathsf{st}(P))$ time.*

Remark. Provided that there exists a constant c (> 1) for which $c^n \cdot \mathsf{tri}(P) \leq \mathsf{st}(P)$ holds for *every* $P \subset \mathbb{R}^2$ of n points, the above running time is dominated by $\mathsf{st}(P)$. It is known that $\mathsf{st}(P)$ becomes minimum when P is in a convex position. On the other hand, $\mathsf{tri}(P)$ is not always minimum for convex positions (see [4]). Furthermore, the number of $\mathsf{st}(P)$ in the convex position is known to be $\Theta(6.75^n)$ [16] relative to the number of triangulations, which is $\Theta(4^n)$, where we ignore polynomial factors. Hence, we strongly conjecture that there exists such a constant $c > 1$.

3.3.2 Enumerating Non-crossing Spanning Connected Graphs

We show here how Algorithm 1 can be applied to the enumeration of non-crossing spanning connected graphs. To efficiently perform Phase 2 of Algorithm 1, we need an algorithm for enumerating all spanning connected subgraphs of a given graph. Although, to the best of our knowledge, previously there was no efficient enumeration algorithm for this graph class, we observe that they can be enumerated in $O(1)$ time per output with $O(n)$ preprocessing time with a slight modification of the algorithm by Uno [33], which was developed for the enumeration of all bases of a matroid (including spanning trees). Let us briefly explain how to modify this algorithm.

The algorithm by Uno is based on the branch-and-bound technique described in the introduction with a balancing operation and a sophisticated amortized analysis. Let $G = (V, E)$ be a given graph. Let us index the edges of E by e_i with $1 \leq i \leq |E|$ in an arbitrary order. Consider enumerating all the spanning trees (i.e., the bases of a graphic matroid) in G by the branch-and-bound technique, starting with a given graph and recursively dividing the problem into two subproblems, where one subproblem is obtained by removing an edge e_i, and the other is obtained by contracting e_i in the ith step. If the graph obtained by removing e_i is disconnected, the algorithm does not proceed further, which is a bounding operation. This algorithm outputs each spanning tree when it reaches a leaf of the branch-and-bound tree. Due to the bounding operation, we easily observe that the algorithm keeps a spanning connected graph during the search. Hence, by outputting graphs not only at leaves but also at some internal nodes, we can enumerate all spanning connected subgraphs of G.

To make this more precise, let us take a look at this branch-and-bound tree in more detail. We can associate a spanning connected subgraph with each of its nodes as follows. The given graph G is associated with the root node of the branch-and-bound tree. Suppose that a node N at depth i has the associated graph G'. Then, N has two children, which have the associated graphs $G' \setminus \{e_i\}$ and G', respectively, if $G' \setminus \{e_i\}$ is connected. Otherwise N has only one child N' which has the associated graph G'. Note that G' is associated with both N and its one child N'. However, since the edge e_i is never removed from the graph at depth greater than i, the edge e_i in G' may be considered as a contracted edge at N'.

[1] The algorithm outputs each graph by the *compact form*, that is, the symmetric difference between the last found object and the current one otherwise it takes $O(n)$ time to output each graph. We remark that the symmetric difference between the last found object and the current one is not necessary of constant size. It can be shown that, if the symmetric difference between two consecutive objects in the search tree (or the branch-and-bound tree) is at most k, then it takes $O(k)$ time to output an object on average (see e.g. [19, 31–33] for more details).

In order to enumerate all the spanning connected subgraphs without repetitions, we initially output G at the root node and then inductively output $G' \setminus \{e_i\}$ after the branching operation at depth i (if $G' \setminus \{e_i\}$ is connected). To see the correctness, let us consider a spanning connected subgraph $G'' = (V, E'')$ of G. Let j be the maximum index among $E \setminus E''$. Then, since G'' can be obtained by removing $E \setminus E''$ from G, there exists a node N'' at depth $j+1$ whose associated graph is G''. It is not difficult to see that a node has the associated graph G'' only if it is either N'' or descendants of N'', and the algorithm outputs G'' only at N''.

To achieve $O(1)$ running time per graph, we just output the symmetric difference of two spanning connected subgraphs which are consecutively output during the enumeration, following the balancing technique proposed by Uno [33]. As a result, the collection of the spanning connected subgraphs can be enumerated in the same time bound as that of the spanning trees.

The edge constraint can be treated easily by edge contraction, and thus all the $R(T)$-constrained spanning connected subgraphs of T can be enumerated in $t_{\mathcal{C}} = O(1)$ time per output with $t_{\mathcal{C},\mathrm{pre}} = O(n)$. Combined with Theorem 3.5, we found that Algorithm 1 enumerates all the non-crossing spanning connected graphs in $O(n \cdot \mathsf{tri}(P) + \mathsf{cg}(P))$ time. Moreover, we obtain the following result.

Theorem 3.7. *For every general point set P in the plane with n points, $1.52^{n-1}\mathsf{tri}(P) \leq \mathsf{cg}(P)$ holds.*

Proof. Let T be a triangulation on P with the minimal representative set $R(T)$. We show that, for every T, there exist at least 1.52^{n-1} non-crossing spanning connected subgraphs in T that are not contained in the other triangulations.

Let us first show that, for every triangle face $p_ip_jp_k$ of T, $|\{(p_i,p_j),(p_i,p_k),(p_j,p_k)\} \cap R(T)| \leq 2$. Without loss of generality, assume that$p_i < p_j < p_k$. Then, notice that the edge (p_j,p_k) is the largest or smallest edge among $\{(p_j,q) \in T \mid q \in P_{j+1}\}$. Hence, (p_j,p_k) is not contained in $R(T)$ by definition of the minimal representative set given in Lemma 3.2.

Consider a subset S of T such that (i) S forms a spanning connected graph on P, (ii) S contains $R(T)$ as its subset, and (iii) S has the minimum edge cardinality among the subsets of T satisfying (i) and (ii). Then, from the above discussion, S contains at most two edges for each face of T. Since S is an $R(T)$-constrained non-crossing spanning connected graph on P, $S \cup F$ forms a distinct $R(T)$-constrained non-crossing spanning connected graph on P for every $F \subseteq T \setminus S$. The number of bounded faces of a triangulation is known to be $2n-h-2$, where h is the number of vertices of the convex hull of P. Since at least one edge of each triangle is contained in $T \setminus S$ and each edge can belong to at most two triangles, by counting the elements of $T \setminus S$ for each triangle, we have $|T \setminus S| \geq (2n-h-2)/2$. Therefore, there exist at least $2^{n-h/2-1}$ subsets of $T \setminus S$, and T contains at least $2^{n-h/2-1}$ $R(T)$-constrained non-crossing spanning connected graphs.

On the other hand (for a point set with large value of h), it can be shown that T contains at least $3^{h/2}$ $R(T)$-constrained non-crossing spanning connected graphs as follows[2]. Notice that no edge of the convex hull of P is contained in $R(T)$, and hence removing arbitrary edges of the convex hull results in an $R(T)$-constrained non-crossing spanning connected graph unless both edges incident to a degree-two vertex of T are removed. If an edge of the convex hull is not incident to a degree-two vertex, there are two possibilities to obtain a connected subgraph of T (i.e., remove it or not). For two edges incident to a degree-two vertex, at most one of them can be removed to obtain a connected graph. Hence there are three possibilities for these two edges (i.e., remove either one of the two edges or leave them). The number of ways to obtain connected subgraphs of T in this manner is minimum if the number of degree-two vertices is maximum, i.e., $h/2$. Thus, T contains at least $3^{h/2}$ $R(T)$-constrained non-crossing spanning connected graphs.

Finally, choosing one of two bounds according to whether $h < \frac{2(n-1)}{1+\log_2 3}$ or not, we obtain the claimed lower bound. □

Theorem 3.7 implies that the running time of Algorithm 1, which is $O(n \cdot \mathsf{tri}(P) + \mathsf{cg}(P))$, is dominated by $\mathsf{cg}(P)$.

[2]This lower bound for the case of large h was pointed out by an anonymous referee.

Theorem 3.8. *Let P be a set of n points in the plane. Then, the set of non-crossing spanning connected graphs on P can be enumerated in $O(\mathsf{cg}(P))$ time.*

3.3.3 Enumerating Plane Straight-line Graphs

For any $F \subseteq T \backslash R(T)$, $F \cup R(T)$ is a plane straight-line graph containing $R(T)$. Hence, by enumerating (the symmetric differences of) all subsets of $T \setminus R(T)$, we can obtain all $R(T)$-constrained plane straight-line graphs in T. Enumerating all subsets of $T \backslash R(T)$ is equivalent to generating all $|T \backslash R(T)|$-bit binary numbers with $O(n)$ preprocessing time, which can be done in constant time per output (see e.g., [29]). Algorithm 1 thus enumerates all the plane straight-line graphs in $O(n \cdot \mathsf{tri}(P) + \mathsf{pg}(P))$ time. Since a non-crossing spanning connected graph is also a plane straight-line graph, $1.52^{n-1}\mathsf{tri}(P) \leq \mathsf{cg}(P) \leq \mathsf{pg}(P)$ holds by Theorem 3.7. Thus, we obtain the following result.

Theorem 3.9. *Let P be a set of n points in the plane. Then, the set of plane straight-line graphs on P can be enumerated in $O(\mathsf{pg}(P))$ time.*

3.3.4 Enumerating Non-crossing Perfect Matchings

Given a point set P of $2n$ points, a *non-crossing perfect matching* is a non-crossing geometric graph on P such that every point of P is incident to exactly one edge of the graph.

Let us consider how to design Phase 2 of Algorithm 1. Suppose that we have an algorithm for finding a perfect matching in a given (non-geometric) graph in $t_{\mathcal{PM}}$ time if it exists. Then, using this algorithm as an oracle, the naively implemented branch-and-bound algorithm can enumerate all the perfect matchings in $O(nt_{\mathcal{PM}})$ time per output graph (see also [28]). The edge constraint can be treated easily. If T has a vertex that is incident to more than one edge of $R(T)$, we report that there is no $R(T)$-constrained perfect matching in T. Otherwise we first remove all edges of $R(T)$ together with the vertices incident to $R(T)$ and then apply the above algorithm for enumerating perfect matchings to the resulting graph. By putting $R(T)$ back to each solution, we obtain all the perfect matchings in T that contain $R(T)$. Algorithm 1 hence enumerates all the non-crossing perfect matchings on P in $O(t_{\mathcal{PM}} \cdot \mathsf{tri}(P) + nt_{\mathcal{PM}} \cdot \mathsf{pm}(P))$ time.

4 Independent Minimal Representative Sets

We know that the algorithm by Bespamyatnikh [12] enumerates all triangulations efficiently, but its search tree is not nicely structured when we focus on the minimal representative sets (see Fig. 5). Namely, for two triangulations T and T' for which T is a parent of T' in the search tree, T' may miss some representative edge that appears in T. Consider, for example, the enumeration of non-crossing matchings. In Phase 2 of Algorithm 1 for a triangulation T, the algorithm outputs no $R(T)$-constrained non-crossing matching if there is a vertex incident to more than one edge of $R(T)$. However, since some descendant triangulation T' of T may not have a vertex which is incident to more than one edge of $R(T')$, T' may contain an $R(T')$-constrained non-crossing matching, and thus we cannot skip the enumeration of T and its descendants. The next proposed algorithm avoids this inefficiency.

We first propose a new algorithm for enumerating triangulations whose search tree has a monotone structure with respect to the minimal representative sets such that $R(T) \subset R(T')$ holds for any triangulation T and its descendant T' (see Fig. 7). Using this monotonicity, we can efficiently enumerate only the minimal representative sets possessing the specified property, which allows us to skip the output of unnecessary triangulations. Let us explain this idea more formally. Recall that $\mathcal{F}$ denotes the collection of all non-crossing edge sets on P. Let $\mathcal{I}$ be a subset of $\mathcal{F}$ satisfying the following independent system;

(I1) $\emptyset \in \mathcal{I}$.

(I2) If $F_2 \in \mathcal{I}$ and $F_1 \subseteq F_2$, then $F_1 \in \mathcal{I}$.

A non-crossing edge set $F \in \mathcal{F}$ is called *independent edge set* or *independent* (with respect to $\mathcal{I}$) if $F \in \mathcal{I}$. If $\mathcal{I}$ satisfies the following condition,

(I3) for every $G \in \mathcal{NGG}$, $G \in \mathcal{I}$ (where G is considered as an edge set),

then we can ensure that the minimal representative set of $T^*(G)$ is independent for every $G \in \mathcal{NGG}$. This implies that it is sufficient to enumerate only the independent minimal representative sets to enumerate all graphs of $\mathcal{NGG}$.

4.1 Enumerating Triangulations Based on Edge Insertions

Our new enumeration algorithm for triangulations is also based on the reverse search [6, 7] whose search tree can be characterized by the root triangulation and the parent-child relation (see Section 3.2 for a brief explanation of the reverse search). Here, we define $T^*(\emptyset)$ as the root triangulation. Hence, the minimal representative set of the root triangulation is empty. For each non-root triangulation T, the parent of T is defined as $T^*(R(T) \setminus \{e\})$ with the smallest edge e among $R(T)$ with respect to the edge ordering $\prec$. The correctness of our parent-child relation follows from the next lemma.

Lemma 4.1. *Let T be a triangulation with $R(T) \neq \emptyset$. Then, for any $e \in R(T)$, the minimal representative set of $T^*(R(T) \setminus \{e\})$ is $R(T) \setminus \{e\}$.*

Proof. Let $T' = T^*(R(T)\setminus\{e\})$. It is sufficient to show $R(T)\setminus\{e\} \subseteq R(T')$ because $R(T') \subseteq R(T)\setminus\{e\}$ by Lemma 3.2.

Consider any $(p_i, p_j) \in R(T) \setminus \{e\}$ with $p_i < p_j$. Let p_ip_jv and p_ip_jw be the two triangles incident to (p_i, p_j) in T, and similarly let p_ip_jv' and p_ip_jw' be those in T'. Without loss of generality, we assume that v and v' (and w and w', resp.) lie on the right side (and the left side, resp.) of (p_i, p_j). Note that $p_i < v$ and $p_i < w$ hold since $(p_i, p_j) \in R(T)$. If $v = v'$ and $w = w'$, then the triangle faces incident to (p_i, p_j) do not differ between T and T'. Hence $(p_i, p_j) \in R(T)$ implies $(p_i, p_j) \in R(T')$ by the definition of the minimal representative set given in Lemma 3.2.

Let us consider the case of $v \neq v'$. When generating $T = T^*(R(T))$ by Construction 1, there exists the cone C with apex p_i which is bounded by (p_i, p_j) and the other consecutive edge among $\delta_{R(T)}(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ to (p_i, p_j) and which contains both p_j and v since $p_i < v$. Let H be the convex hull of $P_{i+1} \cap C$. Then, $v = (p_i, v) \cap H$ holds since $(p_i, v) \in T^*(R(T))$. Similarly, when constructing $T' = T^*(R(T) \setminus \{e\})$, there exists the convex hull H' just below $(p_i, p_j) \in R(T) \setminus \{e\}$ for which $v' = (p_i, v') \cap H'$ holds. Since every vertex visible from p_i with respect to $R(T)$ is still visible from p_i with respect to $R(T) \setminus \{e\}$, all the right endpoints of the edges of $\delta_{R(T)}(p_i) \cup \{(p_i, p_i^{\mathrm{up}}), (p_i, p_i^{\mathrm{low}})\}$ are still visible from p_i with respect to $R(T) \setminus \{e\}$. Thus, $H \subseteq H'$ holds, and hence H' contains v (see Fig. 6).

It is easily observed that, since $H \subseteq H'$, (p_i, p_j) does not become the smallest one among $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$ when removing e (and it is not the largest one either). Hence, by the definition of the minimal representative set, $(p_i, p_j) \in R(T')$ if (p_i, p_j) is flippable in T'. Since there exists no point of P inside the triangle p_ip_jv and no point inside p_ip_jv', either one of the following two cases occurs depending on the position of v': (i) (p_i, v') intersects (v, p_j), or (ii) (v', p_j) intersects (p_i, v). When (i) holds, v' is properly contained in H. However, since $H \subseteq H'$, v' is also properly contained in H', which contradicts $v' = (p_i, v') \cap H'$. Thus, (ii) must hold. In this case the inner angles $\angle p_ip_jv$ and $\angle p_ip_jv'$ satisfy $\angle p_ip_jv' \leq \angle p_ip_jv$. Applying a similar argument to the pair of w and w', we have $\angle p_ip_jw' \leq \angle p_ip_jw$. (However, it is not difficult to see $w = w'$ from the fact that e properly intersects (v', p_j) but not (p_i, p_j).) Hence, the inner angle of the quadrilateral $p_iv'p_jw'$ at p_j is less than π because (p_i, p_j) is flippable in T.

Let us show that the opposite angle, that is, the inner angle of the quadrilateral $p_iv'p_jw'$ at p_i, is also less than π. This can be proved from the fact that both of v' and w' are on the right side of p_i since (p_i, p_j) is neither the smallest nor largest one among $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$ with respect to $\prec$. Hence (p_i, p_j) is flippable in T' and $(p_i, p_j) \in R(T')$ follows. □

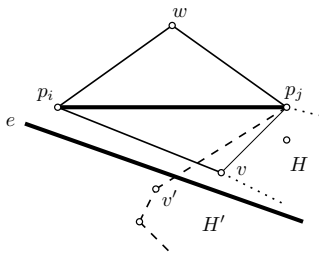


Figure 6: Illustration of the proof of Lemma 4.1, where the bold line represents the removed edge e, the dotted and dashed lines represent the boundaries of H and H', respectively.

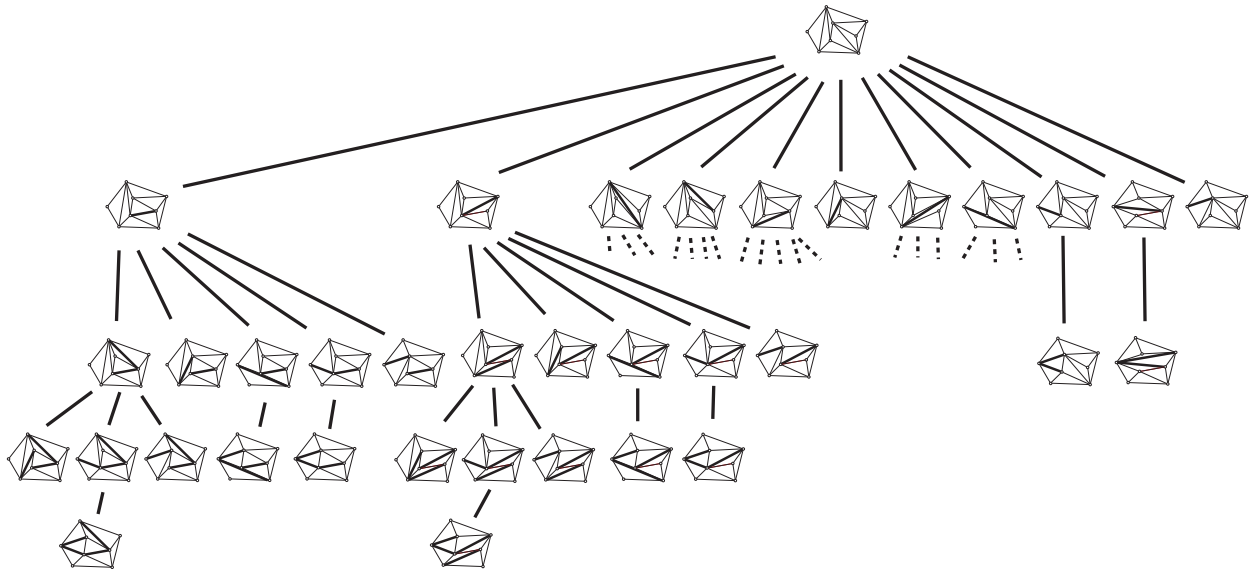


Figure 7: The search tree on the collection of triangulations obtained by the edge-insertion algorithm, where each minimal representative set is drawn in bold.

By Lemma 4.1, $R(T) \subset R(T')$ holds for any triangulation T and its descendant T'. Moreover, since the root triangulation has an empty minimal representative set, our definition of the parent-child relation correctly induces a rooted search tree on the collection of all triangulations. The algorithm traces this search tree in depth-first manner. We call this new algorithm *edge-insertion algorithm* for (enumerating) triangulations. An example of the new search tree is depicted in Fig. 7.

Let us analyze the time complexity of the edge-insertion algorithm. In the reverse search the most time-consuming part is to find all children T' of a triangulation T, i.e., to find all edges $e \in K_n$ for which $T' = T^*(R(T) \cup \{e\})$ is a child of T. Such e can be characterized by the following lemma.

Lemma 4.2. *Let T and T' be triangulations on P for which $T' = T^*(R(T) \cup \{e\})$ holds for some $e \in K_n$, where e does not properly intersect any edge of $R(T)$. Then T' is a child of T if and only if all of the following three conditions are satisfied:*

(a) $e \notin T$,

(b) $e \prec e_1$, *where e_1 is the lexicographically smallest edge among $R(T)$, and*

(c) $R(T) \subseteq R(T')$.

Proof. ("Only-if"-part) Let e' be the lexicographically smallest edge among $R(T')$. Note that by Lemma 4.1 $R(T) = R(T') \setminus \{e'\}$ whenever T is a parent of T' (i.e., $T = T^*(R(T) \setminus \{e'\})$). Hence (c) holds. Also, since $T' = T^*(R(T) \cup \{e\})$, we have $R(T') \subseteq R(T) \cup \{e\}$ by Lemma 3.2. Hence, combining $R(T') \subseteq (R(T') \setminus \{e'\}) \cup \{e\}$ and $e' \in R(T')$, we obtain $e = e'$. Consequently, e is the lexicographically smallest edge among $R(T') = R(T) \cup \{e\}$, implying (b).

Suppose, for a contradiction, that (a) does not hold. Then, we have $R(T) \subseteq R(T) \cup \{e\} \subseteq T$ since $e \in T$, and hence we obtain $R(T) \cup \{e\} \in [T]$ by Lemma 3.2, which implies $T' = T^*(R(T) \cup \{e\}) = T$. This contradicts that T' is a child of T.

("If"-part:) First let us show $e \in R(T')$. Suppose otherwise; then, by the definition of $R(T')$, e is non-flippable in T', or the smallest or largest edge among $\{(p_i, q) \in T' \mid q \in P_{i+1}\}$ for the left endpoint p_i of e. We hence have, by Lemma 2.3, $e \in T' = T^*(R(T) \cup \{e\}) = T^*((R(T) \cup \{e\}) \setminus \{e\}) = T^*(R(T)) = T$, which contradicts condition (a).

Combining $e \in R(T')$ and condition (c), we obtain $R(T) \cup \{e\} \subseteq R(T')$. On the other hand $R(T') \subseteq R(T) \cup \{e\}$ is known from Lemma 3.2. Therefore, $R(T') = R(T) \cup \{e\}$. Condition (b) says that e is the smallest edge among $R(T) \cup \{e\}$, and hence, according to the definition of the parent, $T^*(R(T') \setminus \{e\}) = T^*(R(T)) = T$ is the parent of T'. □

We now concentrate on how to find all edges that produce children of a given triangulation T. We first show that all edges satisfying the conditions of Lemma 4.2 can be found in $O(cn^2)$ time for each T, where c is the subscription of the left endpoint p_c of the smallest edge among $R(T)$ (and c is defined to be n if $R(T) = \emptyset$). Note that the number of edges satisfying condition (b) can be bounded from above by $\sum_{i=1}^{c}(n-i) < cn$. The algorithm checks each of these edges one by one whether it satisfies the other conditions (a) and (c) in $O(n)$ time (per edge). Clearly condition (a) can be checked in $O(n)$ time. To check (c) the algorithm explicitly constructs $T' = T^*(R(T) \cup \{e\})$ in $O(n)$ time based on the method of Lemma 2.4 for each edge e satisfying (a) and (b). Then it is enough to check whether all edges of $R(T)$ are contained in $R(T')$ in T'. This can be done in $O(1)$ time for each edge of $R(T)$ (due to the definition of the minimal representative set given in Lemma 3.2), and thus (c) can be checked in $O(n)$ time. As a result, we can find all edges that satisfy all the conditions of Lemma 4.2 in $O(cn^2)$ time.

This $O(cn^2)$ time is improved to $O(n^2/c)$ time by a simple amortized analysis as follows. Consider the point set $P' = \{p_1, \ldots, p_c\}$. We claim that, for any edge $e \in K_n \setminus T$ whose both endpoints are contained in P', e always satisfies all the conditions of Lemma 4.2. Since such e clearly satisfies (a) and (b) from its definition, let us confirm that e also satisfies (c). Notice that every edge of $R(T)$ lies completely to the right side of the right endpoint of e due to the definition of p_c. Hence, inserting e into $R(T)$ does not affect the right side of p_c when constructing $T^*(R(T) \cup \{e\})$, i.e., every $e' \in R(T)$ is incident to the same two triangles in $T^*(R(T) \cup \{e\})$ as in $T = T^*(R(T))$, and all edges of $R(T)$ are still contained in the minimal representative set of $T^*(R(T) \cup \{e\})$. Thus, e satisfies (c).

The number of edges $e \in K_n \setminus T$ whose both endpoints are contained in P' is at least $c(c-1)/2 - (3c-6)$. This implies that there exist $\Omega(c^2)$ children of T. Distributing the time $O(cn^2)$ evenly to $\Omega(c^2)$ children and T itself, we obtain the result.

Theorem 4.3. *Let P be a set of n points. Then, the edge-insertion algorithm enumerates all the triangulations on P in $O(n^2)$ time per output graph without repetition.*

4.2 Enumerating Independent Minimal Representative Sets

Owing to the nicely structured search tree of the minimal representative sets, we can now perform the efficient enumeration of the independent minimal representative sets (defined at the beginning of Section 4) and the corresponding triangulations.

Algorithm 2: Enumeration of $\mathcal{NGG}$.

Phase 1: Execute the edge-insertion algorithm starting from $T^*(\emptyset)$ as described in Section 4.1 to enumerate triangulations.

Phase 2: Every time a new triangulation T is found, check whether $R(T)$ is independent or not. If $R(T)$ is dependent, skip the enumeration of all the descendants of T.

Phase 3: Every time a new independent $R(T)$ is found, enumerate all $R(T)$-constrained graphs of $\mathcal{NGG}$ in T.

The correctness of Algorithm 2 follows from the next lemma.

Lemma 4.4. *Let $\mathcal{I}$ be the collection of independent edge sets of $\mathcal{F}$. Then, Algorithm 2 correctly enumerates all graphs of $\mathcal{NGG}$ without repetitions if $\mathcal{I}$ satisfies (I1), (I2), and (I3).*

Proof. We first note that all of the independent minimal representative sets are correctly enumerated in Algorithm 2. To verify this, let us imagine the search tree which is obtained by performing the edge-insertion algorithm for enumerating triangulations. The subgraph of this search tree induced by all T with $R(T) \in \mathcal{I}$ forms a rooted tree by (I1) and (I2), and hence the algorithm enumerates every independent $R(T)$ correctly.

Let us show that every $G \in \mathcal{NGG}$ is actually enumerated. Lemma 3.2 states $R(T^*(G)) \subseteq G \subseteq T^*(G)$. Since $G \in \mathcal{I}$ holds by (I3), $R(T^*(G)) \in \mathcal{I}$ follows from (I2). Thus, G is enumerated in Phase 3 for $T^*(G)$. □

Let us analyze the time complexity of Algorithm 2 under the assumption that $\mathcal{I}$ satisfies (I1), (I2), and (I3). Assume that there exists an oracle that checks in t_{check} time whether $I \cup \{e\} \in \mathcal{I}$ or not for an independent set I and an edge $e \in K_n$. Let $\mathcal{I}_{\mathrm{rep}} \subseteq \mathcal{I}$ be the collection of the independent minimal representative sets on a given point set P. We can easily observe that the time to be spent in Phase 1 and 2 is $O(n^2 \cdot t_{\mathrm{check}} \cdot |\mathcal{I}_{\mathrm{rep}}|)$ since there exist $O(n^2)$ children for each triangulation on the search tree and from Theorem 4.3. Hence, using the notation $\mathcal{C}, t_{\mathcal{C}}$ and $t_{\mathcal{C},\mathrm{pre}}$ defined in Theorem 3.5, we obtain the following result:

Theorem 4.5. *Algorithm 2 enumerates all the graphs of $\mathcal{NGG}$ on a given point set P without repetitions in $O((n^2 \cdot t_{\mathrm{check}} + t_{\mathcal{C},\mathrm{pre}}) \cdot |\mathcal{I}_{\mathrm{rep}}| + t_{\mathcal{C}} \cdot \mathsf{ngg}(P))$ time. Moreover, the time complexity is bounded by $O((n^2 \cdot t_{\mathrm{check}} + t_{\mathcal{C},\mathrm{pre}} + t_{\mathcal{C}}) \cdot \mathsf{ngg}(P))$, which is polynomial on average, if $|\mathcal{I}_{\mathrm{rep}}| \leq \mathsf{ngg}(P)$.*

4.3 Application of Algorithm 2

We show here how Algorithm 2 can be applied to the enumeration of non-crossing minimally rigid frameworks. A graph $G = (V, E)$ is *minimally rigid* if $|E| = 2|V| - 3$ and every subgraph of G induced by $V' \subseteq V$ spans at most $2|V'| - 3$ edges. An embedded minimally rigid graph on a planar (generic) point set is called *(generically) minimally rigid framework*. It is known that the collection of minimally rigid frameworks forms a *rigidity matroid* defined on the edge set of K_n (see, e.g., [17]).

We define the *independence* on $\mathcal{F}$ in such a way that $F \in \mathcal{F}$ is independent if and only if F is independent in the rigidity matroid on K_n. Then, since the edge set of each minimally rigid framework is a base of the rigidity matroid, the collection $\mathcal{I}$ of the independent edge sets of $\mathcal{F}$ satisfies (I1), (I2) and (I3). To bound $|\mathcal{I}_{\mathrm{rep}}|$, we remark the following known fact:

Lemma 4.6. ([9]) *Let F be a non-crossing edge set on P that is an independent set in the rigidity matroid on K_n. Then every F-constrained triangulation on P contains an F-constrained minimally rigid framework.*

Hence, a triangulation T contains at least one $R(T)$-constrained non-crossing minimally rigid framework if $R(T)$ is independent, which implies $|\mathcal{I}_{\mathrm{rep}}| \leq \mathsf{mrf}(P)$.

Let us consider the time complexity of Phase 2 of Algorithm 2. For a graph $G = (V, I)$ with n vertices and an independent set I of the rigidity matroid, a maximal rigid subgraph $G' = (V', I')$ of G (i.e., a subgraph with the maximal subset $I' \subseteq I$ satisfying $|I'| = 2|V'| - 3$), is called a *rigid component*. Then $I \cup \{e\}$ is independent if and only if both endpoints of e do not belong to the same rigid component. It is known that all the rigid components of G can be detected in $O(n^2)$ time [11, 27]. Moreover, using the data structure by Lee and Streinu [27] or Berg and Jordán [11] that maintains rigid components, it can be checked in $O(1)$ time whether two vertices belong to the same rigid component. Thus, the algorithm can check in $t_{\mathrm{check}} = O(1)$ time whether the minimal representative set of a new child, that is, $R(T) \cup \{e\}$, is independent or not. If $R(T) \cup \{e\}$ is independent, the algorithm enters Phase 3 while updating the rigid components in $t_{\mathrm{update}} = O(n)$ time for each edge insertion [11, 27]. Algorithm 2 hence enumerates all the independent minimal representative sets (and the corresponding triangulations) in $O(n^2 \cdot t_{\mathrm{check}} + t_{\mathrm{update}}) = O(n^2)$ time per output.

Next, let us consider Phase 3. We use the algorithm by Uno [33] for enumerating all the bases of a matroid. Given a matroid $\mathcal{M}$ on a ground set E with rank r, the algorithm generates all bases of $\mathcal{M}$ in $t_{\mathcal{C}} = O(t_{\mathrm{cir}}/r)$ time per base with the preprocessing time $t_{\mathcal{C},\mathrm{pre}}$, where t_{cir} is the time to calculate the fundamental circuit of $B \cup \{e\}$ for a base B and $e \in E \setminus B$, and $t_{\mathcal{C},\mathrm{pre}}$ is the time to compute the coloops of the matroid in E (where $e \in E$ is called a coloop if all bases contain e). In the case of the rigidity matroid, the algorithm by Berg and Jordán [11] can detect the circuit of $B \cup \{e\}$ in $t_{\mathrm{cir}} = O(r^2)$ time. Moreover, they also developed an algorithm for detecting all the coloops in E in $t_{\mathcal{C},\mathrm{pre}} = O(r^2)$ time. Since the rank r of the rigidity matroid is at most $2n-3$, it thus enumerates all the minimally rigid graphs in G that contain a specified edge set in $t_{\mathcal{C}} = O(t_{\mathrm{cir}}/r) = O(n)$ time per output graph with $t_{\mathcal{C},\mathrm{pre}} = O(n^2)$ preprocessing time. Putting these facts and Theorem 4.5 together gives the following result:

Theorem 4.7. *Let P be a set of n points in the plane. Then the set of non-crossing (generically) minimally rigid frameworks on P can be enumerated without repetitions in $O(n^2 \cdot \mathsf{mrf}(P))$ time.*

This result improves the previous one by [8], which requires $O(n^3)$ time per graph. We note that Algorithm 1 enumerates all non-crossing minimally rigid frameworks in $O(n^2 \cdot \mathsf{tri}(P) + n \cdot \mathsf{mrf}(P))$ time.

5 Other Applications

We proposed a new algorithmic framework for the efficient enumeration of non-crossing geometric graphs, and by applying our technique we obtained the improved algorithms for several specific graph classes. We briefly show below applications of the proposed framework to further graph classes. Algorithm 1 always works in time proportional to the number of triangulations and objects to be enumerated. Whereas, in some problems, Algorithm 2 works practically faster than Algorithm 1, although it seems a nontrivial task to evaluate its running time theoretically.

Non-crossing red-and-blue matchings: For a given point set P, every point is assumed to have either red or blue color. A non-crossing red-and-blue matching is a non-crossing matching on P each of whose edges is not allowed to connect points of the same color. The enumeration can be performed by using the algorithm for enumerating the matchings in a (non-geometric) bipartite graph [35] in Phase 2 of Algorithm 1 or in Phase 3 of Algorithm 2, which needs $t_{\mathcal{C}} = O(n)$ time per output with $t_{\mathcal{C},\mathrm{pre}} = O(n^{3/2})$ preprocessing time (if the edge cardinality of a given graph is $O(n)$). Hence, by Theorem 3.5, Algorithm 1 enumerates all non-crossing red-and-blue matchings in $O(n^{3/2} \cdot \mathsf{tri}(P) + n \cdot \mathsf{rbm})$ time, where rbm is the total number of non-crossing red-and-blue matchings on P, which depends not only on P but also on the coloring of each point.

Algorithm 2 can enumerate all the red-and-blue matchings efficiently if we define $\mathcal{I}$ as the collection of F such that no two edges of F are incident to a vertex and no edge of F connects points of the same color. Notice that every independent minimal representative set is also a non-crossing red-and-blue matching, which implies $|\mathcal{I}_{\mathrm{rep}}| \leq \mathsf{rbm}$. The independence of each non-crossing edge set is trivially checked in $t_{\mathrm{check}} = O(1)$ time, and thus Algorithm 2 works in $O(n^2 \cdot \mathsf{rbm})$ time by Theorem 4.5.

Non-crossing k-vertex or k-edge connected graphs: A non-crossing k-vertex (or k-edge) connected graph is a non-crossing geometric graph spanning a given point set P that remains connected after removing any $k-1$ vertices (or $k-1$ edges) from the graph. Since it can be checked in a polynomial time Q_k whether a given (non-geometric) graph is k-vertex connected (or k-edge connected) or not, according to the branch-and-bound technique discussed in the introduction, we can enumerate k-vertex connected (or k-edge connected) subgraphs in $t_{\mathcal{C}} = O(mQ_k)$ time per output with $t_{\mathcal{C},\mathrm{pre}} = O(n+m+Q_k)$ preprocessing time, where m denotes the number of edges in a subgraph. Thus, using this algorithm in Phase 2, Algorithm 1 enumerates all non-crossing k-vertex (or k-edge)

connected graphs in $O((n+Q_k)\cdot \mathsf{tri}(P)+nQ_k\cdot \mathsf{cg}_k(P))$ time, where $\mathsf{cg}_k(P)$ denotes the total number of non-crossing k-vertex (or k-edge) connected graphs on P.

In particular, it is known that 2-vertex (or 2-edge) connectivity of a graph can be checked in linear time (see, e.g., [30, Chapter 15.2b]). Moreover, $\mathsf{tri}(P) \leq \mathsf{cg}_2(P)$ for every point set P since every triangulation is also a non-crossing 2-vertex (or 2-edge) connected graph on P. Algorithm 1 hence enumerates all the non-crossing 2-vertex (or 2-edge) connected graphs in $O(n^2\mathsf{cg}_2(P))$ time.

Non-crossing directed spanning trees: Each edge of the given geometric complete graph on P is assumed to have an orientation. A non-crossing directed spanning tree (or non-crossing r-arborescence) is a non-crossing spanning tree on P having a unique directed path from a rooted point r to all points of $P\setminus\{r\}$. The enumeration can be performed by using the algorithm of [20, 34] in Phase 2 of Algorithm 1, or in Phase 3 of Algorithm 2. Given a digraph D whose number of arcs is $O(n)$, this algorithm enumerates all the directed spanning trees in D in $t_{\mathcal{C}} = O(\log^2 n)$ time per graph with $t_{\mathcal{C},\mathrm{pre}} = O(n\log n)$ preprocessing time. Hence, Algorithm 1 works in $O(n\log n\cdot \mathsf{tri}(P)+\log^2 n\cdot \mathsf{dst})$ time, where dst denotes the total number of the non-crossing directed spanning trees which depends not only on P but also on the orientation of D.

Algorithm 2 can enumerate all the non-crossing directed spanning trees if we define $\mathcal{I}$ as the collection of the non-crossing edge sets F such that F has no cycle and no vertex has indegree more than one in the directed graph induced by F, then clearly $t_{\mathrm{check}} = O(1)$. Its running time becomes $O(n^2\cdot|\mathcal{I}_{\mathrm{rep}}|+\log^2 n\cdot \mathsf{dst})$.

Edge-constrained non-crossing geometric graphs: The technique can be also applied to the enumeration of S-constrained non-crossing geometric graphs that are those containing a given specified edge set S as their subsets, e.g., S-constrained non-crossing spanning trees or S-constrained non-crossing matchings. This is because both the algorithm by Bespamyatnikh [12] and the edge-insertion algorithm proposed in this paper for enumerating triangulations can be naturally extended to those for enumerating only the S-constrained triangulations by restricting the collection of non-crossing edge sets $\mathcal{F}$ to those containing S as their subsets.

For Algorithm 1, the S-constrained triangulations can be enumerated in $O(\log\log n)$ time per output (see [21]), while the edge-insertion algorithm for Algorithm 2 enumerates them in $O(n^3)$ time per output by setting the root as $T^*(S)$ instead of $T^*(\emptyset)$. (Note that we cannot use an amortized analysis as done in the proof of Theorem 4.3 to achieve an $O(n^2)$ bound.) Efficient algorithms for enumerating edge-constrained non-crossing spanning trees and edge-constrained minimally rigid frameworks are proposed in [21] and [9], respectively, which are based on the reverse search technique.

Acknowledgments

We would like to thank the referees for a lot of helpful comments. One of the anonymous referees pointed out how to improve the result of Theorem 3.7 from $\frac{1.41^n}{2}\mathsf{tri}(P) \leq \mathsf{cg}(P)$ to $1.52^{n-1}\mathsf{tri}(P) \leq \mathsf{cg}(P)$ by a simple idea. We greatly appreciate this comment. The first author is supported by the project *New Horizons in Computing*, Grant-in-Aid for Scientific Research on Priority Areas, MEXT Japan, and by Grant-in-Aid for Scientific Research (C), JSPS. The second author is supported by Grant-in-Aid for JSPS Research Fellowship for Young Scientists.

References

[1] O. Aichholzer, F. Aurenhammer, C. Huemer and H. Krasser, Transforming spanning trees and pseudo-triangulations. *Inf. Process. Lett.*, 97(1), 19–22 (2006).

[2] O. Aichholzer, F. Aurenhammer, C. Huemer, and B. Vogtenhuber, Gray code enumeration of plane straight-line graphs. *Graphs and Combinatorics*, 23(5), 467–479 (2007).

[3] O. Aichholzer, F. Aurenhammer and F. Hurtado, Sequences of spanning trees and a fixed tree theorem. *Comput. Geom.*, 21(1–2), 3–20 (2002).

[4] O. Aichholzer, T. Hackl, C. Huemer, F. Hurtado, H. Krasser, and B. Vogtenhuber, On the number of plane geometric graphs. *Graphs and Combinatorics*, 23(1), 67–84 (2007).

[5] O. Aichholzer and K. Reinhardt, A quadratic distance bound on sliding between crossing-free spanning trees. *Comput. Geom. Theory Appl.*, 37, 155–161 (2007).

[6] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8, 295–313 (1992).

[7] D. Avis and K. Fukuda, Reverse search for enumeration. *Discrete Appl. Math.*, 65(1–3), 21–46 (1996).

[8] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa, Enumerating non-crossing minimally rigid frameworks, *Graphs and Combinatorics*, 23(1), 117–134 (2007).

[9] D. Avis, N. Katoh, M. Ohsaki, I. Streinu and S. Tanigawa, Enumerating constrained non-crossing minimally rigid frameworks. *Discrete Comput. Geom.*, 40(1), 31–46 (2008).

[10] S. Bereg, Enumerating pseudo-triangulations in the plane. *Comput. Geom. Theory Appl.*, 30(3), 207–222 (2005).

[11] A. Berg and T. Jordán, Algorithms for graph rigidity and scene analysis. In *Proc. 11th Annual European Symposium on Algorithms (ESA)*, pp. 78–89, Lecture Notes in Computer Science, vol. 2832, Springer-Verlag, (2003).

[12] S. Bespamyatnikh, An efficient algorithm for enumeration of triangulations. *Comput. Geom. Theory Appl.*, 23(3), 271–279 (2002).

[13] H. Brönnimann, L. Kettner, M. Pocchiola and J. Snoeyink, Counting and enumerating pointed pseudo-triangulations with the greedy flip algorithm *SIAM J. Comput.*, 36(3), 721–739 (2006).

[14] B. Chazelle, Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5), 485–524 (1991).

[15] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to Algorithms. Second Edition, The MIT Press (2001).

[16] P. Flajolet and M. Noy, Analytic combinatorics of non-crossing configurations. *Discrete Math.*, 204, 203–229 (1999).

[17] J. Graver, B. Servatius, and H. Servatius, Combinatorial Rigidity. Graduate Studies in Mathematics vol. 2. American Mathematical Society (1993).

[18] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir and R. E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2, 209–233 (1987).

[19] S. Kapoor and H. Ramesh, Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM J. Comput.*, 24(2), 247–265 (1995).

[20] S. Kapoor and H. Ramesh, An algorithm for enumerating all spanning trees of a directed graph. *Algorithmica*, 27(2), 120–130 (2000).

[21] N. Katoh and S. Tanigawa, Enumerating edge-constrained triangulations and edge-constrained non-crossing spanning trees. *Discrete Appl. Math.*, (to appear).

[22] M. C. Hernando, M. E. Houle and F. Hurtado, On local transformation of polygons with visibility properties. In *Proc.6th International Conference Computing and Combinatorics,COCOON 2000*, pp. 54–63, Lecture Notes in Computer Science, vol. 1858, Springer-Verlag (2000).

[23] C. Hernando, F. Hurtado and M. Noy, Graphs of non-crossing perfect matchings. *Graphs and Combinatorics*, 18(3), 517–532 (2002).

[24] M. E. Houle, F. Hurtado, M. Noy and E. Rivera-Campo, Graphs of triangulations and perfect matchings. *Graphs and Combinatorics*, 21(3), 325-331 (2005).

[25] F. Hurtado, M. Noy and J. Urrutia, Flipping edges in triangulations. *Discrete Comput. Geom.*, 22(3), 333–346 (1999).

[26] K. Jansen and G. J. Woeginger, The complexity of detecting crossingfree configurations in the plane. *BIT*, 33(4), 580-595 (1993).

[27] A. Lee and I. Streinu, Pebble game algorithms and sparse graphs. *Discrete Math.*, 308(8), 1425–1437 (2008).

[28] Y. Matsui, T. Matsui and K. Fukuda, A catalog of enumeration algorithms. `http://roso.epfl.ch/kf/enum/enum.html`.

[29] C. Savage, A survey of combinatorial Gray codes. *SIAM Review*, 39, 605–629 (1997).

[30] A. Schrijver, Combinatorial Optimization. Polyhedra and Efficiency. Springer-Verlag, Heidelberg (2003).

[31] A. Shioura and A. Tamura, Efficiently scanning all spanning trees of an undirected graph. *Journal of the Operations Research Society of Japan*, 38(3), 331–344, The Operations Research Society of Japan (1995).

[32] A. Shioura, A. Tamura and T. Uno, An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3), 678–692 (1997).

[33] T. Uno, A new approach for speeding up enumeration algorithms and its application for matroid bases. In *Proc. 5th Computing and Combinatorics Conference (COCOON '99)*, pp. 54–63, Lecture Notes in Computer Science, vol. 1627, Springer-Verlag (1999).

[34] T. Uno, A new approach for speeding up enumeration algorithms. In *Proc. 9th International Symposium on Algorithm and Computation (ISAAC '98)*, pp. 287–296, Lecture Notes in Computer Science, vol. 1533, Springer-Verlag (1998).

[35] T. Uno, Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In *Proc. 8th International Symposium on Algorithms and Computation (ISAAC '97)*, pp. 92–101, Lecture Notes in Computer Science, vol. 1350, Springer-Verlag (1997).

[36] E. Welzl, Counting of crossing-free geometric graphs. *Significant Advances in Computer Science (SACS 07)*, Graz, Austria. Nov. (2007).